

Comp 411 Notes - Fri, April 25, 2005

Absolute and Existential Types: Part 1

Scribe: Seth J. Fogarty

This lecture concerns the encoding of types like `val := Int of int | Fun of val -> val` and the motivation for existential typing.

1 Typing of val

Recall that `val` has the church encoding $\forall X. (\text{Nat} \rightarrow X) \rightarrow ((X \rightarrow X) \rightarrow X) \rightarrow X$.

We were able to construct `int` as $\lambda i : \text{int}. \lambda X. \lambda f i : \text{int} \rightarrow X. \lambda ff : (X \rightarrow X) \rightarrow X. f i i$

We attempted to construct `fun`. `Fun` would have start with

$\lambda f : \text{val} \rightarrow \text{val}. \lambda X \lambda f i : \text{int} \rightarrow X. \lambda ff : (X \rightarrow X) \rightarrow X$.

There are two possibilities for what we could from here.

1. Apply some interesting value to f another value v_2 . Use this value v_2 , X , $f i$, and ff to create an X .
2. Somehow apply f to ff , which is in the style of `int`. This would require us to make a function $\lambda x : X \dots$ that would return something of type X and wrap around f .

We were unable to solve this problem in class.

2 Existential Types

Since we have types that correspond to the logical universal quantifier, we will consider types that correspond to the logical existential quantifier. These types $\exists N.t$ read “There exists some type N , such that the term has type t .”

An immediate question is the possibly correspondence between absolute and existential types. In logic the proposition $\exists A.t$ is identical to the proposition $\neg \forall A. (\neg t)$. Unfortunately, we have no conception of not in our type system.

One way of saying $\neg t$ is to say that $t \rightarrow \text{false}$. In logic, that t implies false. In type theory, that there is some function from a value of type t to a value of type `false`. Unfortunately, we have no conception of false in our type system.

Recall that a type is a proposition, and a term represents a proof of that proposition. Thus we would like `false` to be an uninhabited typed. The simplest such type is $\forall A.A$. A term of this type would be very useful. You give me a type, and I will give you a value of that type. Further, since System F programs always terminate, that would mean that I really would get a value of that type. Thus it is a reasonable assertion that this type is uninhabited, and would make an adequate false.

So we can then translate $\neg t$ to $t \rightarrow \forall A.A$. As a reality check, consider `¬false`. This is the type $\forall A.A \rightarrow \forall A.A$, and it is inhabited by $\lambda x : \forall A.A.x$. Thus the identity function is a proof of true.

Under this translation, $\exists A.t$ is equivalent to $T1 = (\forall X.(t \rightarrow \forall A.A)) \rightarrow \forall A.A$.

This is slightly different from the usual translation, which is $T2 = \forall Z.((\forall X.t \rightarrow Z) \rightarrow Z)$.

Naturally we want to see if there is an isomorphism between these two expressions of the existential quantifier. An isomorphism would be proved if we could find a context, or a function, that would translate any term of one type to any term of the other. Translating T2 to T1 is easy: $\lambda x.x[\forall A.A]$. This function takes anything of T2 and produces it to something of T1, and is a proof that T2 implies T1.

We do not know how to get from T1 to T2. This is a very interesting problem to consider.

We can also consider an alternate form of not. $\neg t = \forall X.t \rightarrow X$. This is the type that, given any X, I can give you a function from t to X. If X is false, you get the more direct notion of *not*.

This idea of *not* seems closer than our original, but the version of $\exists N.T$ that it results in is $\forall Z.((\forall X.\forall Y.t \rightarrow Y) \rightarrow Z)$. It is not directly obvious that this corresponds in any way to the usual translation of the existential type.